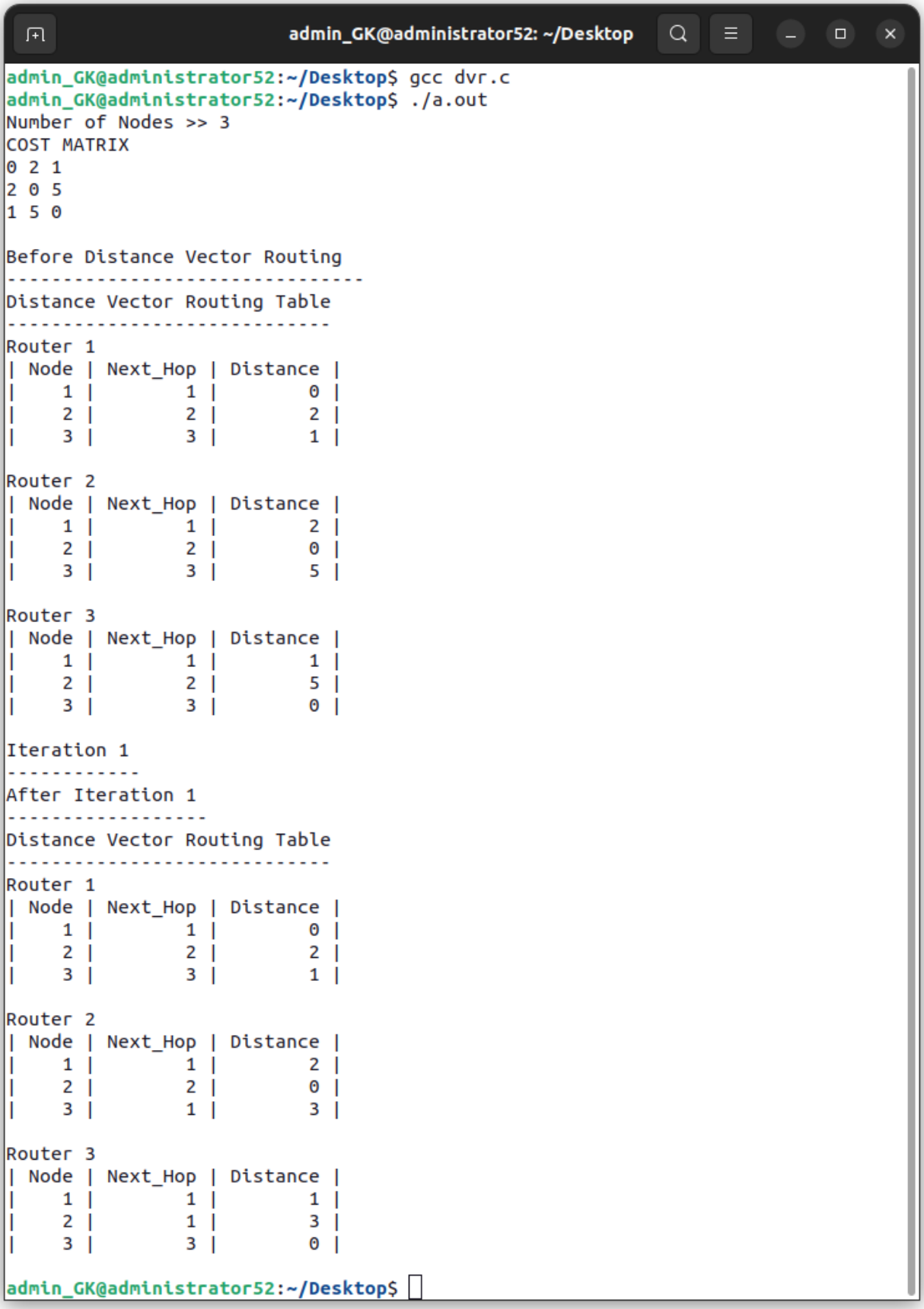
Program

```
#include <stdio.h>
#define MAX_NODES 20
#define INFINITY 999999

int costMatrix[MAX_NODES][MAX_NODES], n;

struct routers
{
    int distance[MAX_NODES];
    int adjNodes[MAX_NODES];
} node[MAX_NODES];

// Function to display the distance vector routing table
void displayRoutingTable()
{
    int i, j;
    printf("Distance Vector Routing Table\n");
    printf("-----------------------------\n");
    for (i = 0; i < n; ++i)
    {
        printf("Router %d\n", i + 1);
        printf("| Node | Next_Hop | Distance |\n");
        for (j = 0; j < n; ++j)
        {
            printf("| %4d | %8d | %8d |\n", j + 1, node[i].adjNodes[j] + 1,
node[i].distance[j]);
        }
        printf("\n");
    }
}

// Function to read the cost matrix
void readCostMatrix()
{
    int i, j;
    printf("COST MATRIX\n");
    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            scanf("%d", &costMatrix[i][j]);
            // Distance from node X to itself is 0
            if (i == j)
                costMatrix[i][j] = 0;
            // Initialize distance vector and adjacent node array
            node[i].distance[j] = costMatrix[i][j];
            node[i].adjNodes[j] = j;
        }
    }
}

// Function to update distance vectors based on Bellman-Ford algorithm
void updateDistanceVectors()
{
    int i, j, k;
    for (k = 0; k < n-2; ++k)
```

```
    {
        printf("Iteration %d\n", k + 1);
        printf("------------\n");
        for (i = 0; i < n; ++i)
        {
            for (j = 0; j < n; ++j)
            {
                // Update distance vector using Bellman-Ford algorithm
                if (node[i].distance[j] > costMatrix[i][k] + node[k].distance[j])
                {
                    node[i].distance[j] = costMatrix[i][k] + node[k].distance[j];
                    node[i].adjNodes[j] = k;
                }
            }
        }
        // Display distance vector after each iteration
        printf("After Iteration %d\n", k + 1);
        printf("------------------\n");
        displayRoutingTable();
    }
}

int main()
{
    printf("Number of Nodes >> ");
    scanf("%d", &n);
    readCostMatrix();
    // Before distance vector routing table
    printf("\nBefore Distance Vector Routing\n");
    printf("------------------------------\n");
    displayRoutingTable();
    // Update distance vectors using Bellman-Ford algorithm
    updateDistanceVectors();
    return 0;
}
```

Output


admin_GK@administrator52: ~/Desktop
admin_GK@administrator52:~/Desktop$ gcc dvr.c
admin_GK@administrator52:~/Desktop$ ./a.out
Number of Nodes >> 3
COST MATRIX
0 2 1
2 0 5
1 5 0
Before Distance Vector Routing
Distance Vector Routing Table
Router 1
Node	Next_Hop	Distance
1	1	0
2	2	2
3	3	1
Router 2
Node	Next_Hop	Distance
1	1	2
2	2	0
3	3	5
Router 3
Node	Next_Hop	Distance
1	1	1
2	2	5
3	3	0
Iteration 1
After Iteration 1
Distance Vector Routing Table
Router 1
Node	Next_Hop	Distance
1	1	0
2	2	2
3	3	1
Router 2
Node	Next_Hop	Distance
1	1	2
2	2	0
3	1	3
Router 3
Node	Next_Hop	Distance
1	1	1
2	1	3
3	3	0
admin_GK@administrator52:~/Desktop$